MU4RBI01 – Projet *Python*

ELION GALIBA Fady, NOCHÉ Kévin & SIVATHASAN Ramya

15 décembre 2024

Introduction

Ce présent document a pour but d'expliquer comment le projet python de l'UE *MU4RBI01* de *Sorbonne Université*, département scientifique, a été travaillé par le groupe n°5 de la formation d'IPS-TSDM (auteur de ce rapport). Pour rappel, le but de ledit projet est de créer un simple jeu de tactique, mettant en scène au moins deux joueurs, dans lequel ces derniers s'affrontent avec des unités (équivalent de soldats), tout en utilisant, dans le code source, des classes avec des liens spécifiques entre elles (relation mère-fille, etc).

Le projet a été réalisé grâce à *Github*, via l'intermédiaire de *Git*. Les logiciels tels que *Kate*, *Konsole* ou *Spyder* ont été essentiels au bon déroulement du projet.

Le repository du projet peut être retrouvé via ce site[1].

Fonctionnalités du jeu

Résumé

Dans le jeu, deux équipes s'affrontent : les dragons du bien (à droite de l'écran) et les dragons du mal (à gauche de l'écran). Les dragons du mal étant « méchants » (Quoique ?...), c'est à eux de commencer.

Le jeu se joue un peu comme aux échecs : chaque joueur, durant son tour, fait agir une seule unité. Ici, chaque unité ne peut effectuer qu'une seule action par tour (à l'exception du *Gueux* – voir plus bas) : se déplacer, attaquer ou effectuer sa capacité spéciale.

Dans chaque équipe, on retrouve trois types d'unité, identiques pour chaque équipe :

— Le *Royal* : C'est l'unité la plus puissante. Elle a une vitesse de 2 (c'est-à-dire qu'elle peut se déplacer de deux cases par tour), une puissance d'attaque de 32 (elle retire 32 points de vie lors d'une attaque simple, moins la résistance de l'unité attaquée), a une résistance de 16 (chaque attaque contre elle a un malus de 16 points de dégâts, avec un minimum de zéro) et a une santé de 60 (la santé maximum). Elle représente le chef d'état, dirigeant (peu importe l'équipe) d'une poigne de fer ses troupes.

— Le *Soldat* : C'est une unité également très puissante en raison de son attaque spéciale, que nous verrons plus bas. Elle a une vitesse de 3, une résistance de 13, une attaque de 23 et une santé de 36. Dans le jeu, on lui associe l'« aura politique » autour du *Royal*, ainsi que l'immunité des élites politiques.

— Le *Gueux* : Cela se voit à son nom misérable, il s'agit de l'unité représentant le peuple. Avec une vitesse de 2, une résistance de 11, une puissance d'attaque de 17 et une santé de 23, il s'agit de l'unité la moins puissante. En revanche, il s'agit également de la seule unité capable de faire toutes ses actions en un tour (se déplacer, attaquer et/ou lancer sa capacité spéciale, dans n'importe quel ordre) ; elle est donc non négligeable.

Chaque unité a une capacité spéciale. Voici ci-dessous une liste de leurs pouvoirs :

— Le *Royal* peut, après avoir cibler une unité, effectuer une attaque berserk contre cette unité adverse. L'attaque berserk inflige 1,5 fois les dégâts ordinaires du *Royal*, tout en supprimant la résistance de l'unité attaquée. En revanche, ce pouvoir n'est pas sans contre-partie ; après avoir attaqué, le *Royal* reçoit la moitié de sa puissance d'attaque en dommages plus 16, c'est-à-dire la moitié de son attaque tout en supprimant sa propore résistance aux dégâts (étant de 16).

— Le *Soldat* peut, lorsque sa capacité spéciale est activée, cibler une case. S'il attaque cette case ciblée, il inflige sa puissance d'attaque moins 4 (soit 23 – 6 = 17 d'attaque) à l'unité présente sur cette case. Il inflige

[1] **URL**
https://github.com/NKevinVI/Sorbonne_SdI_IPS_TSDM_MU4RBI01_Project

autant de dégâts aux unités adjacentes à la case initialement touchée (sans compter les diagonales).

— Le *Gueux* a une technique bien à lui qui peut lui permettre de subsister plus longtemps sur le terrain : il peut se régénérer. Il regagne la moitié de sa puissance d'attaque en tant que points de vie, sans pouvoir cependant dépasser sa jauge maximale de santé, qui est celle fixée au tout début de la partie.

Règles et Mode d'Emploi

Jouer sans avoir regardé le manuel d'utilisation équivaut à essayer d'utiliser un ordinateur sans périphériques. Voici donc, pour votre plus grand bonheur (ou malheur, suivant contre qui vous jouez), les règles du jeu *Draconic Generations* :

Touche(s)	Action effectuée.
Clique gauche Souris	Permet de choisir l'unité à jouer. Une fois l'unité sélectionnée, on joue sans retour en arrière !
Touches directionnelles	Pour déplacer l'unité que vous avez choisie.
`Z`, `Q`, `S`, `D`	Permettent de cibler une case adjacente, en vue d'attaquer en aval.
Espace	Une fois l'attaque simple dirigée sur une unité, permet d'attaquer.
`X`	Action spéciale de l'unité.
Échap	Dans certains cas, annule le coup spéciale de l'unité.
Ctrl	Rafraîchit la fenêtre (d'autres boutons peuvent fonctionner, mais on conseille à ceux allergiques à l'informatique d'utiliser celui-ci).

Comme indiqué précédemment, c'est le joueur « du mal » qui commence.

Fonctionnalités supplémentaires

- Du mana peut être récupéré afin d'améliorer la puissance d'attaque de chaque unité (et la capacité de régénération du *Gueux*).
- Au bout de soixante tours sans aucune mort, le jeu se termine sur une égalité des joueurs.
- Il est possible de relancer le jeu à la fin d'une partie.
- Le jeu possède un *Easter Egg*, seul cas où les deux joueurs gagnent simultanément. Si les soldats sont éliminés dans chaque camp, qu'aucun gueux n'a été tué et que les royaux sont côte-à-côte, alors les deux royaumes draconiques se lient, et la partie se termine en une victoire des deux joueurs (il n'existe aucun autre moyen de faire remporter la victoire aux deux joueurs en même temps).
- Il est possible d'agrandir la fenêtre de jeu en la plaçant tout en haut de l'écran, ce qui l'agrandit automatiquement. Il n'est cependant pas possible de l'agrandir confortablement en l'étirant avec le curseur.

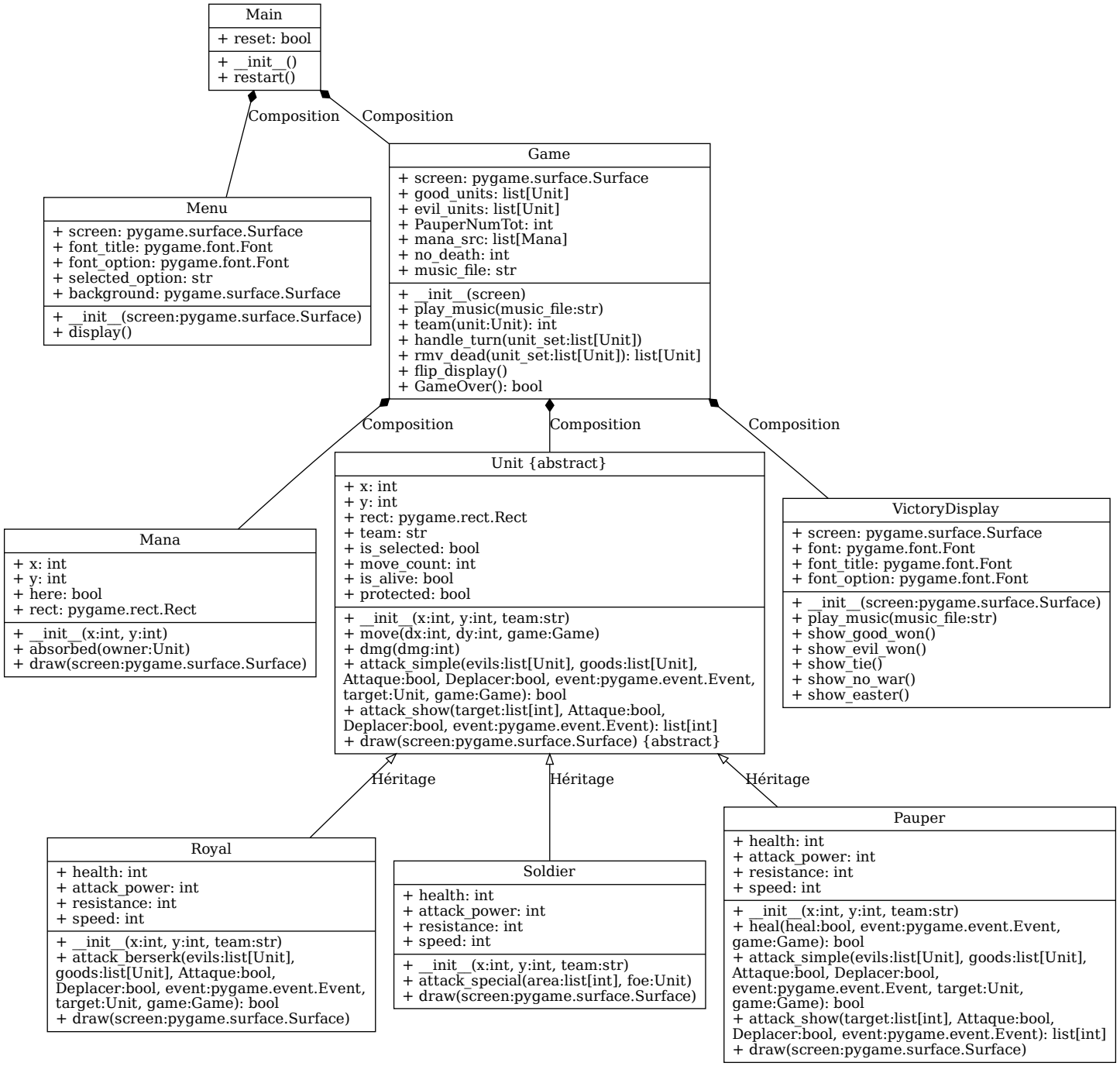
Organisation du Projet

Pour ce qui est de la séparation des tâches, Fady s'est principalement occupé des fichiers `VictoryDisplay.py` et `menu.py`, permettant ainsi d'avoir une interface de début et de fin de session de jeu ; il a également joué un rôle majeur pour ce qui est de l'ajout de musique au sein du jeu. Kévin s'est occupé de la classe `unit.py`, `game.py` et `main.py`, en créant un nouveau fichier `var.py` pour y placer toutes les variables ; nous y retrouvons ainsi tout le cœur central du jeu, avec la gestion d'une partie complète entre deux joueurs, ainsi que la possibilité de rejouer une fois la partie terminée. Ramya a pu faire des mises à jour importantes dans le code, notamment dans les fichiers `mana.py` ou `menu.py`, et a travaillé sur l'affichage de la grille du jeu, dans le fichier `game.py`. Avec Fady, ils se sont beaucoup occupé du diagramme UML.

Sur le Diagramme UML

La classe *Unit* est abstraite car nous l'utilisons pour créer nos classes *Pauper* (*Gueux*), *Soldier* (*Soldat*) et *Royal*, qui sont toutes des unités. De ce fait, *Unit* est la classe mère de ces trois classes citées précédemment.

Toutes les autres classes du jeu sont imbriquées les unes dans les autres (*VictoryDisplay* est utilisée par *Game* qui est elle-même utilisée par *Main*). Ainsi, nous trouvons beaucoup de relation de composition, les unités importantes étant composées d'autres unités moins complexes (relations fortes).


Main
+ reset: bool
+ __init__()
+ restart()
Composition
Composition
Menu
+ screen: pygame.surface.Surface
+ font_title: pygame.font.Font
+ font_option: pygame.font.Font
+ selected_option: str
+ background: pygame.surface.Surface
+ __init__(screen:pygame.surface.Surface)
+ display()
Game
+ screen: pygame.surface.Surface
+ good_units: list[Unit]
+ evil_units: list[Unit]
+ PauperNumTot: int
+ mana_src: list[Mana]
+ no_death: int
+ music_file: str
+ __init__(screen)
+ play_music(music_file:str)
+ team(unit:Unit): int
+ handle_turn(unit_set:list[Unit])
+ rmv_dead(unit_set:list[Unit]): list[Unit]
+ flip_display()
+ GameOver(): bool
Composition
Composition
Composition
Mana
+ x: int
+ y: int
+ here: bool
+ rect: pygame.rect.Rect
+ __init__(x:int, y:int)
+ absorbed(owner:Unit)
+ draw(screen:pygame.surface.Surface)
Unit {abstract}
+ x: int
+ y: int
+ rect: pygame.rect.Rect
+ team: str
+ is_selected: bool
+ move_count: int
+ is_alive: bool
+ protected: bool
+ __init__(x:int, y:int, team:str)
+ move(dx:int, dy:int, game:Game)
+ dmg(dmg:int)
+ attack_simple(evils:list[Unit], goods:list[Unit], Attaque:bool, Deplacer:bool, event:pygame.event.Event, target:Unit, game:Game): bool
+ attack_show(target:list[int], Attaque:bool, Deplacer:bool, event:pygame.event.Event): list[int]
+ draw(screen:pygame.surface.Surface) {abstract}
VictoryDisplay
+ screen: pygame.surface.Surface
+ font: pygame.font.Font
+ font_title: pygame.font.Font
+ font_option: pygame.font.Font
+ __init__(screen:pygame.surface.Surface)
+ play_music(music_file:str)
+ show_good_won()
+ show_evil_won()
+ show_tie()
+ show_no_war()
+ show_easter()
Héritage
Héritage
Héritage
Royal
+ health: int
+ attack_power: int
+ resistance: int
+ speed: int
+ __init__(x:int, y:int, team:str)
+ attack_berserk(evils:list[Unit], goods:list[Unit], Attaque:bool, Deplacer:bool, event:pygame.event.Event, target:Unit, game:Game): bool
+ draw(screen:pygame.surface.Surface)
Soldier
+ health: int
+ attack_power: int
+ resistance: int
+ speed: int
+ __init__(x:int, y:int, team:str)
+ attack_special(area:list[int], foe:Unit)
+ draw(screen:pygame.surface.Surface)
Pauper
+ health: int
+ attack_power: int
+ resistance: int
+ speed: int
+ __init__(x:int, y:int, team:str)
+ heal(heal:bool, event:pygame.event.Event, game:Game): bool
+ attack_simple(evils:list[Unit], goods:list[Unit], Attaque:bool, Deplacer:bool, event:pygame.event.Event, target:Unit, game:Game): bool
+ attack_show(target:list[int], Attaque:bool, Deplacer:bool, event:pygame.event.Event): list[int]
+ draw(screen:pygame.surface.Surface)